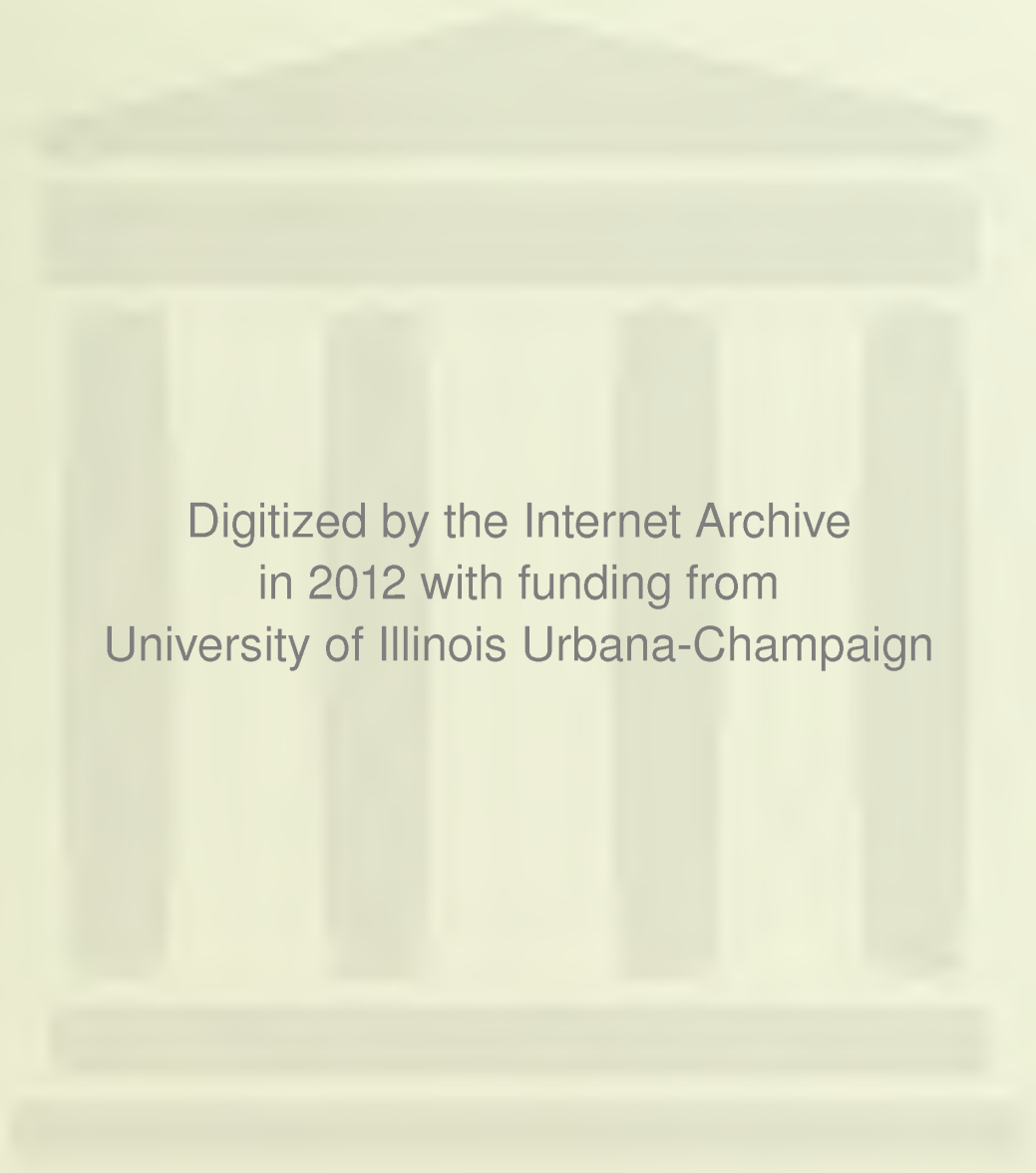






UNIVERSITY OF  
ILLINOIS LIBRARY  
AT URBANA-CHAMPAIGN  
BOOKSTACKS



Digitized by the Internet Archive  
in 2012 with funding from  
University of Illinois Urbana-Champaign

<http://www.archive.org/details/gradientbasedlea164horn>



THE LIBRARY OF THE  
NOV 11 1993  
UNIVERSITY OF ILLINOIS  
URBANA-CHAMPAIGN

# Gradient-Based Learning in Recurrent Networks

*Kurt Hornik*

*Institut für Statistik und Wahrscheinlichkeitstheorie  
Technische Universität Wien*

*Chung-Ming Kuan*

*Department of Economics  
University of Illinois*



# BEBR

FACULTY WORKING PAPER NO. 93-0164

College of Commerce and Business Administration

University of Illinois at Urbana-Champaign

September 1993

## Gradient-Based Learning in Recurrent Networks

Kurt Hornik

Chung-Ming Kuan





# Gradient-Based Learning in Recurrent Networks

**Kurt Hornik**

Institut für Statistik und Wahrscheinlichkeitstheorie  
Technische Universität Wien

**Chung-Ming Kuan**

Department of Economics  
University of Illinois at Urbana-Champaign

September 18, 1993



## Abstract

We discuss gradient-based learning in recurrent neural networks. The basic equations are derived in a general framework of continuous-time dynamical systems; from these, the well-known discrete-time recurrent backpropagation algorithms are deduced. The convergence properties of such algorithms are analyzed; in particular, it is emphasized that constraints on the recurrent weights have to be enforced to ensure proper convergence.

**Keywords:** Additive model; Contraction mapping; Elman network; Jordan network; ODE method, Recurrent backpropagation algorithm; Recurrent network; Recurrent Newton algorithm.





# 1 Introduction

Recurrent networks differ from feedforward networks in that internal feedback connections among units are permitted. On the one hand, recurrent networks embed richer dynamic structures so that they are able to capture more temporal characteristics of the target sequence, cf. e.g. Jordan (1986). On the other hand, recurrent networks summarize and store past information compactly in recurrent units so that they are capable of performing cognition even when inputs are static (Norrod, O'Neill, & Gat, 1987). Thus, recurrent networks are particularly useful for applications in which temporal structure plays an important role. For such and many other reasons, recurrent networks have been the subject of broad research interest within the last few years. The fact that the IEEE Transactions on Neural Networks have devoted one of their 1993 issues exclusively to this topic is a clear indication of the importance of these systems.

In this paper, we discuss some well known gradient-based learning algorithms for recurrent networks and their convergence properties. The basic equations for gradient descent are derived in a general framework of continuous-time dynamical systems in section 2. From these, the discrete-time recurrent backpropagation (BP) algorithms are obtained in section 3; a Newton-type algorithm is introduced as well. The convergence properties of these learning rules are discussed in section 4. In particular, it is emphasized that there are constraints to the feedback connections which should be imposed during the learning process to ensure proper convergence.

## 2 Basics

The idea of adjusting the parameters of a (general nonlinear) dynamical system by gradient descent on some performance functional is neither new nor specific to the neural network (NN) field. Such methods have successfully been employed in systems theory and control for a considerable amount of time (although the emphasis in these areas has clearly been on linear systems). Nevertheless, in the sequel we shall provide a rather self-contained discussion of the basic ideas and refer to the neural networks literature only.

For sake of notational simplicity we shall base our derivations on continuous-time systems; discrete-time models are discussed in section 3. Hence, consider a dynamical system of the form

$$\dot{u} = f(t, u; w) \tag{1}$$

with initial condition  $u(t_0) = u_0$ . Here,  $u$  is the  $\nu$ -dimensional vector of state variables,

and  $w$  is a  $\mu$ -dimensional vector of parameters (“weights”) which is to be adjusted to achieve certain “suitable” behavior of the system. The exact form of  $f$  is irrelevant for the derivation of the basic equations, but may heavily influence their actual implementation. In typical NN applications, the  $i$ -th component  $f_i(t, u; w)$  of  $f$  is of the form  $f_i(u_i, \text{net}_i, x_i)$ , where  $\text{net}_i = \sum_j w_{ij} u_j$  and  $x_i$  are the net and external inputs to unit  $i$  at time  $t$ , respectively,  $w_{ij}$  is the  $(i, j)$ -th element of the matrix of network interconnection strengths  $W$ , and the parameter vector  $w$  contains the elements of  $W$ . E.g., we could use  $w = \text{vec}(W)$ , where the  $\text{vec}$  operator stacks one column of a matrix underneath the other. In particular, in (one version of) the standard additive model,

$$\dot{u}_i = -u_i + \sigma_i(\text{net}_i) + x_i, \quad (2)$$

where  $\sigma_i$  is a squashing activation function. In what follows, we shall use lower and upper case letters for vectors and matrices, respectively. Subscripts denote the corresponding entries.

The performance of the system could be measured in several different ways. In some applications, it may depend on the whole trajectory  $(u(t), t_0 \leq t \leq t_f)$  for some “final”  $t_f$ ; in others, only the states  $u(t_k)$  at certain discrete times  $t_k$  or even  $u(t)$  in the limit for  $t \rightarrow \infty$  may be of interest. We shall refer to these cases as “trajectory-based” and “state-based”, respectively. Notice that in the NN literature “trajectory learning” is typically contrasted to “fixed point learning”. But clearly, the latter concept only adequately characterizes the situations where performance is measured in terms of a static equilibrium reached by the system for  $t \rightarrow \infty$ , as is appropriate when e.g. training a network to be an associative memory, but not the state-based cases where such an equilibrium is not reached (e.g., if the inputs come from some stationary random process) or finite time horizons are of interest (e.g., in language recognition). Of course, combinations between state-based and trajectory-based criteria are also possible; e.g. in a typical control application, the goal might be to reach a desired final state with as little cost as possible.

Let  $\ell(t) = \ell(t, u(t))$  be the performance of the system at time  $t$ . In typical cases,  $\ell(t)$  is the instantaneous (prediction) error

$$\ell(t) = \frac{1}{2} \sum_{i \in O(t)} (u_i(t) - y_i(t))^2,$$

where  $O(t)$  and  $y(t)$  are the set of output units and the target, respectively, employed at time  $t$ . Note that  $u$  and hence also  $\ell$  are of course functions of the adjustable parameters  $w$  as well, although this dependence is not made notationally explicit. Similarly, we shall

drop other arguments to functions when no confusion arises by doing so. To see how  $\ell$  varies with  $w$ , we can use the chain rule to obtain

$$\frac{\partial \ell}{\partial w_j} = \sum_i \frac{\partial \ell}{\partial u_i} \frac{\partial u_i}{\partial w_j},$$

and, using (1),

$$\frac{\partial}{\partial t} \frac{\partial u_i}{\partial w_j} = \sum_k \frac{\partial f_i}{\partial u_k} \frac{\partial u_k}{\partial w_j} + \frac{\partial f_i}{\partial w_j}.$$

In what follows, it will be convenient to write  $\partial g / \partial v$  for the Jacobian  $[\partial g_i / \partial v_j]$ , where  $i$  and  $j$  are the row and column indices, respectively, and  $\nabla_v g$  for the gradient  $(\partial g / \partial v)'$ . Hence, if  $\ell$  is a scalar function,  $\partial \ell / \partial w$  is a row vector and  $\nabla_w \ell$  is a column vector. Using this notation, we have

$$\frac{\partial \ell}{\partial w} = \frac{\partial \ell}{\partial u} S, \tag{3}$$

where  $S = \partial u / \partial w$  satisfies the inhomogeneous linear ODE

$$\dot{S} = \frac{\partial f}{\partial u} S + \frac{\partial f}{\partial w}. \tag{4}$$

The matrix  $S$  represents how the state  $u$  varies with infinitesimal changes in  $w$ . In control theory,  $S$  is often called the *sensitivity matrix*, and equation (4) is referred to as the *forward sensitivity equation*.

In gradient descent, the parameters  $w$  are updated proportional to the negative gradient of the performance functional  $\Lambda$  to be minimized, i.e.,  $\Delta w = -\eta \nabla_w \Lambda$ , where  $\eta$  is the learning rate. Hence, in the state-based case with  $\Lambda = \ell(t_f)$  (updating at  $t_f$ ) we have

$$\Delta w = -\eta S(t_f)' \nabla_u \ell(t_f).$$

In the trajectory-based cases, performance is typically measured by a functional

$$\Lambda = \int_{t_0}^{t_f} \ell(t) dt. \tag{5}$$

Hence, gradient descent modifies  $w$  according to

$$\Delta w = -\eta \int_{t_0}^{t_f} \nabla_w \ell(t) dt = -\eta \int_{t_0}^{t_f} S(t)' \nabla_u \ell(t) dt. \tag{6}$$

Notice that in the latter case we could also accumulate the weight changes with time using  $\dot{w} = -\eta S' \nabla_u \ell$ .

Equation (6) is just a (continuous-time and more generally written) version of the recurrent backpropagation algorithm as proposed e.g. by Robinson & Fallside (1987),

Williams & Zipser (1989a, b). A variant with exponentially weighted instantaneous errors is discussed in Gherrity (1989). If equation (5) is discretized in a way that weight updates are performed along with each integration step of equation (4), we obtain the real-time recurrent backpropagation algorithm of Williams & Zipser, see also Kuan, Hornik, & White (1993).

The above shows that the exact gradients could always be computed forward in time based on a direct integration of the forward sensitivity system. Nevertheless, they might also be computed or approximated differently; in the sequel, we shall discuss some examples.

As in neural network applications,  $\mu$  typically is of order  $\nu^2 \gg \nu$ , a direct integration of the forward sensitivity equations is rather costly (in addition, we observe that we basically solve the same system for each column of  $S$ ). Alternatively, we may proceed as follows. Let  $F(t)$  be a fundamental solution of the associated homogeneous system

$$\dot{F} = \frac{\partial f}{\partial u} F, \quad F(t_0) = I,$$

and write  $K(t, \tau)$  for the transport matrix  $F(t)F(\tau)^{-1}$ . Then by the well-known solution formula for linear inhomogeneous system with non-constant coefficients, we obtain that for all  $t_0 \leq s \leq t$ ,

$$S(t) = K(t, s)S(s) + \int_s^t K(t, \tau) \frac{\partial f}{\partial w}(\tau) d\tau. \quad (7)$$

In particular, if the initial value  $u(t_0)$  does not depend on  $w$ ,  $S(t_0) = 0$ , hence  $S(t) = \int_{t_0}^t K(t, \tau) \frac{\partial f}{\partial w}(\tau) d\tau$  and

$$\frac{\partial \ell}{\partial w}(t) = \frac{\partial \ell}{\partial u}(t) F(t) \int_{t_0}^t F(\tau)^{-1} \frac{\partial f}{\partial w}(\tau) d\tau.$$

As  $G(t) = F(t)^{-1}$  solves the system  $\dot{G} = -G \frac{\partial f}{\partial u}$ , we can, as proposed in Sun, Chen & Lee (1992), set up two auxiliary systems

$$\begin{aligned} \dot{G} &= -G \frac{\partial f}{\partial u}, & G(t_0) &= I, \\ \dot{H} &= G \frac{\partial f}{\partial w}, & H(t_0) &= 0, \end{aligned}$$

and use

$$\frac{\partial \ell}{\partial w}(t) = a(t)H(t),$$

where  $a(t)$  solves the linear equation  $a(t)G(t) = \partial \ell / \partial u(t)$ . In typical NN applications where  $\mu = O(\nu^2)$  and  $\partial f / \partial w$  only contains  $O(\nu^2)$  nonzero elements, this requires only  $O(\nu^3)$  operations per time step, as opposed to  $O(\nu^4)$  for the direct integration.



In the state-based case with  $t_f = \infty$ , it may not be necessary to follow the dynamics of the forward sensitivity equations. In fact, assume that all quantities of interest converge for  $t \rightarrow \infty$  (the case of “fixed point learning”), we find that  $S(\infty)$  solves the equation

$$\frac{\partial f}{\partial u}(\infty)S(\infty) + \frac{\partial f}{\partial w}(\infty) = 0,$$

i.e.,  $S(\infty) = -(\partial f/\partial u(\infty))^{-1} \partial f/\partial w(\infty)$ , and hence

$$\frac{\partial \ell}{\partial w}(\infty) = -\frac{\partial \ell}{\partial u}(\infty) \left( \frac{\partial f}{\partial u}(\infty) \right)^{-1} \frac{\partial f}{\partial w}(\infty).$$

(Notice however that, as pointed out e.g. in Pearlmutter (1990), the mapping  $w \mapsto u(\infty; w)$  is not necessarily continuous for all  $w$  of interest, even for globally convergent systems. For such  $w$ , the argument clearly cannot be applied.) Of course, the efficient way of computing the above expression is to first obtain the solution  $v(\infty)$  of the (“small”) linear system  $v \partial f/\partial u(\infty) = \partial \ell/\partial u(\infty)$  and then compute  $-v(\infty) \partial f/\partial w(\infty)$ .

To use this relation, one could first run the system (1) up to some  $t_f$  which is large enough to get  $u(t_f)$  close to the equilibrium  $u(\infty)$  (if this is asymptotically stable, convergence is exponentially fast) and then approximate  $v(\infty)$  by the solution of  $v \partial f/\partial u(t_f) = \partial \ell/\partial u(t_f)$ . When applied to (discrete-time) feedforward architectures where convergence occurs after a finite number of time steps, this reduces to ordinary backpropagation, as the linear system can then be solved directly by backsubstitution due to the block triangularity of  $\partial f/\partial u$ .

Alternatively,  $v(\infty)$  could be computed by a relaxation method as the equilibrium of the system

$$\dot{v} = v \frac{\partial f}{\partial u} + \frac{\partial \ell}{\partial u}. \quad (8)$$

In this case, the approximate gradients are computed by strict forward propagation. When applied to the additive model (2), one obtains exactly the recurrent backpropagation algorithm discussed in Pineda (1987). Almeida (1987) gives a similar algorithm for a slightly different model.

In the trajectory-based case, we can utilize the explicit solution formula (7) of the forward sensitivity equations (again assuming that  $S(t_0) = 0$ ) to obtain by a simple change of the order of integration that

$$\frac{\partial \Lambda}{\partial w} = \int_{t_0}^{t_f} \frac{\partial \ell}{\partial u}(t) \left( \int_{t_0}^t K(t, s) \frac{\partial f}{\partial w}(s) ds \right) dt = \int_{t_0}^{t_f} v(t) \frac{\partial f}{\partial w}(t) dt,$$

where

$$v(t) = \int_t^{t_f} \frac{\partial \ell}{\partial u}(s) K(s, t) ds$$

is the solution of

$$\dot{v} = -v \frac{\partial f}{\partial u} - \frac{\partial \ell}{\partial u}$$

with the *final* condition  $v(t_f) = 0$ . This is sometimes referred to as the *adjoint* (or backwards) *sensitivity system*.

Hence in this case, we can also compute the gradient by first running the system (1) forward in time and then computing  $v(t)$  along with  $\int_t^{t_f} v(s) \partial f / \partial w(s) ds$  backwards in time. When applied to the standard additive model (2), this method gives the algorithm suggested by Pearlmutter (1989), cf. also Toomarian & Barhen (1991). When applied to corresponding discrete-time systems, one obtains the “backpropagation through time” algorithm of Rumelhart, Hinton & Williams (1986), and  $v(t)$  can be interpreted as the vector of back-propagated signals. For more details, see Baldi (1993).

One potential drawback of this method is that the integration backwards in time makes it necessary to store the whole trajectory  $u(t)$  for  $t_0 \leq t \leq t_f$  before the computation of  $v$  and  $\partial \ell / \partial w$  can begin. This need for a potentially infinitely large stack can be avoided by solving the adjoint sensitivity equations forward in time along with an auxiliary system. Let  $\tilde{v}(t) = \int_{t_0}^t \partial \ell / \partial u(s) K(s, t) ds$  be the solution of the adjoint system with zero initial condition. Then clearly,  $v(t) = -\tilde{v}(t_f) K(t_f, t) + \tilde{v}(t)$ . Hence,

$$\frac{\partial \Lambda}{\partial w} = -\tilde{v}(t_f) F(t_f) \int_{t_0}^{t_f} F(t)^{-1} \frac{\partial f}{\partial w}(t) dt + \int_{t_0}^{t_f} \tilde{v}(t) \frac{\partial f}{\partial w}(t) dt.$$

Clearly, both integrals on the right-hand side can be accumulated forward in time and combined at time  $t_f$  with the first one premultiplied by  $-\tilde{v}(t_f) F(t_f)$ ; however, it is not clear whether this can be implemented in a way which is more efficient than a direct forward propagation based on the forward sensitivity equations.

Among the above algorithms, none is uniformly better than all others for all possible applications. We have already seen that backpropagation through time scales poorly with the number of time steps (the pattern lengths) before updating. On the other hand, its single time steps are typically quicker than those of algorithms based on forward propagation. For a more detailed discussion of this complexity issue, see e.g. Baldi (1993).

### 3 Examples and Extensions

Thus far, gradient-based learning has been studied explicitly in the framework of continuous time dynamical systems only. Equivalently, we could study discrete-time systems of the form

$$u_{t+1} = g(t, u_t; w). \tag{9}$$

There is a one-to-one correspondence between (1) and (9). If we discretize (1) on a grid of mesh  $\Delta$  and write  $\tilde{u}_k = u(t_0 + k\Delta)$ , we obtain

$$\tilde{u}_{k+1} \approx \tilde{u}_k + \Delta f(t_0 + k\Delta, \tilde{u}_k; w) := g(k, \tilde{u}_k; w).$$

In particular, if  $t_0 = 0$  and  $\Delta = 1$ , we have the correspondence  $g(k, u; w) = u + f(k, u; w)$ . Notice however that this correspondence is only formal; the two systems can behave quite differently unless the unit of time for which  $\Delta = 1$  is small enough to make difference ratios good approximations to derivatives.

As a first example, consider the fully-connected discrete time model

$$u_{t+1} = \sigma(Au_t + Bx_t) \tag{10}$$

considered in Williams & Zipser (1989a, b), where  $\sigma$  performs componentwise squashing, i.e.,  $\sigma(v) = [\sigma_i(v_i)]$ . Writing  $W = [A, B]$ ,  $z = [u', x']'$  and  $\text{net} = Au + Bx = Wz$ , we can more compactly write this system as  $u_{t+1} = \sigma(\text{net}_t)$ . Letting  $w = \text{vec}(W)$ , the corresponding continuous-time system is  $\dot{u} = -u + \sigma(\text{net}) = f(t, u; w)$ , for which

$$\frac{\partial f}{\partial u} = -I + D\sigma(\text{net})A, \quad \frac{\partial f}{\partial w} = z' \otimes D\sigma(\text{net}),$$

where  $D\sigma_i$  is the ordinary derivative of  $\sigma_i$ ,  $D\sigma$  is the matrix with diagonal entries  $D\sigma_i$  and zero off-diagonal entries, and  $\otimes$  denotes the Kronecker product. (For arbitrary matrices  $P$  and  $Q$ , their Kronecker product  $P \otimes Q$  is obtained from  $P$  by replacing each entry  $p_{ij}$  of  $P$  with the matrix  $p_{ij}Q$ . Notice that if  $z$  is a column vector, then  $P(z' \otimes I) = (1 \otimes P)(z' \otimes I) = z' \otimes P$ , where the last step follows from the product rule for Kronecker products; for more details, cf. e.g. Magnus & Neudecker (1988).)

Assume for simplicity that all units are visible and compared to targets  $y$  at each time step. With the usual quadratic performance measure  $\ell_t(u) = (1/2) \sum_i (y_{i,t} - u_i)^2$ , we have  $\partial \ell_t / \partial u = -e'_t$ , where  $e_t = y_t - u_t$  is the network “error” at time  $t$ . For the corresponding continuous-time system, we obtain

$$\begin{aligned} \frac{\partial \ell}{\partial w} &= -e'_t S, \\ \dot{S} &= -S + D\sigma(\text{net})(AS + z'_t \otimes I). \end{aligned}$$

Discretizing this forward sensitivity system with mesh  $\Delta = 1$ , we thus obtain the discrete-time system

$$S_{t+1} = D\sigma(\text{net}_t)(AS_t + z'_t \otimes I),$$

which is exactly the recurrent backpropagation algorithm of Williams & Zipser. Componentwise, the above reads as

$$s_{kl,t+1}^i = D\sigma_i(\text{net}_{i,t}) \left( \sum_j w_{ij} s_{kl,t}^j + \delta_{ik} z_{l,t} \right),$$

where  $s_{kl,t}^i$  is  $\partial u_i / \partial w_{kl}$  at time  $t$ , and  $\delta_{ik}$  the Kronecker delta. Extensions which include teacher forcing can be obtained similarly.

The above system can be used to implement a variety of different weight updating schemes. In particular, we can run the system from time  $t_0$  to  $t_f$  and use  $\Delta w = \eta \sum_{t_0+1}^{t_f} S'_t e_t$  (the trajectory-based case) or  $\Delta w = S'_{t_f} e_{t_f}$  (the state-based case). A special case is obtained by updating the weights at each time step (i.e.,  $t_f = t_0 + 1$ ). This gives the approximate gradient descent scheme

$$\begin{aligned} \hat{e}_t &= y_t - \hat{u}_t, \\ \hat{w}_{t+1} &= w_t + \eta_t \hat{S}'_t \hat{e}_t, \\ \hat{S}_{t+1} &= D\sigma(\widehat{\text{net}}_t)(\hat{A}_t \hat{S}_t + \hat{z}'_t \otimes I), \end{aligned}$$

where here and in what follows a variable is written with a “hat” symbol if it is evaluated at the parameter estimates  $\hat{w}$ . This is just the *real-time* recurrent backpropagation algorithm of Williams & Zipser.

As a second example, let us apply the idea of fixed point learning to the discrete time system (10). The relaxation equation (8) for the corresponding continuous-time system is

$$\dot{v} = -v + v D\sigma(\text{net})A - e',$$

which discretizes into

$$v_{t+1} = v_t D\sigma(\text{net}_t)A - e'_t.$$

This gives the approximation

$$\frac{\partial \ell_\infty}{\partial w} \approx v_t \frac{\partial f_t}{\partial w} = v_t (z'_t \otimes D\sigma(\text{net}_t)) = z'_t \otimes v_t D\sigma(\text{net}_t),$$

which componentwise reads as  $\partial \ell_\infty / \partial w_{kl} \approx v_{k,t} D\sigma_k(\text{net}_{k,t}) z_{l,t}$ . This is a hebbian rule with the presynaptic activity term  $z$  and the postsynaptic term  $-v D\sigma(\text{net})$ ; of course, some special machinery is required to compute the latter. This interpretation might suggest a modification which is based on a relaxation method for the equilibrium value  $p_\infty = -v_\infty D\sigma(\text{net}_\infty)$  of the postsynaptic term. Observing that  $p_\infty$  solves the linear equation  $p = (pA + e'_\infty) D\sigma(\text{net}_\infty)$ , we can approximate it through

$$p_{t+1} = (p_t A + e'_t) D\sigma(\text{net}_t)$$



and use

$$\frac{\partial \ell_\infty}{\partial w} \approx -z'_t \otimes p_t.$$

An on-line version based on these fixed-point approximations with weight updates at each time step is thus

$$\begin{aligned}\hat{e}_t &= y_t - \hat{u}_t, \\ \hat{W}_{t+1} &= \hat{W}_t + \eta_t \hat{z}_t \hat{p}_t, \\ \hat{p}_{t+1} &= (\hat{p}_t \hat{A}_t + \hat{e}_t) D\sigma(\widehat{\text{net}}_t),\end{aligned}$$

which is the recurrent backpropagation algorithm of Pineda (1987).

A more general system than (10) is studied in Kuan, Hornik & White (1993). Their starting points are networks with a single hidden layer and delayed internal feedbacks. Such networks can be described by the equations

$$\begin{aligned}u_{t+1} &= g(u_t, x_t; w) \\ o_t &= h(C\sigma(Au_t + Bx_t)),\end{aligned}$$

where  $w = [\text{vec}(A)', \text{vec}(B)', \text{vec}(C)']'$  is the vector of network connection strengths. Clearly,  $\sigma(Au_t + Bx_t)$  is the vector of hidden unit activations at time  $t$ . When  $g \equiv 0$  such that  $u_t = 0$ , the above reduces to a single hidden layer feedforward network. If  $u_t = o_{t-1}$ , we obtain the Jordan network with output feedbacks, cf. Jordan (1986, 1992); if  $u_t = \sigma(Au_{t-1} + Bx_{t-1})$  is the lagged vector of hidden unit activations, we obtain the Elman network with hidden-unit activations feedbacks, see Elman (1990). If  $h = \text{id}$ ,  $C = I$  and  $u_t = o_{t-1}$  (such that the system and output equations are identical), we obtain the model (10) considered earlier, which could thus also be termed a Jordan network with hidden and output layer collapsed.

The above class of discrete time neural networks is particularly attractive in applications in dynamic environments. For example, Jordan (1986, 1992) is concerned with the control and learning of robot movements; hence the Jordan network was originally constructed so that it can “memorize” previous positions of the robot. More generally, the Jordan network is particularly suitable when the serial order of outputs is important. On the other hand, Elman (1990) is interested in learning sequential pattern in linguistics, such as the syntactic or semantic features of words; the Elman network was then constructed to “memorize” internal states, i.e., hidden unit activations. Thus, hidden-unit activation feedbacks in Elman networks in fact encode the temporal properties of sequential inputs; this is extremely useful in many dynamic applications.

The exact functional form of the output equations is not important for the derivation of the gradient computations. Hence, more generally, consider a system  $u_{t+1} = g(u_t, x_t; w)$  along with a general output equation

$$o_t = h(u_t, x_t; w).$$

The basic updating equations could now easily be obtained by applying the general principle to an augmented state vector  $\tilde{u}_t = [o'_{t-1}, u'_t]'$  (note that the time indexing differs from the previously considered models). Alternatively, and more conveniently, we can proceed as follows. If the performance measure  $\ell_t = \ell_t(o_t)$  is a function of the outputs which in turn are computed from the states of the system *and* the system's parameters, we have

$$\frac{\partial \ell}{\partial w} = \frac{\partial \ell}{\partial o} \frac{\partial o}{\partial w} = \frac{\partial \ell}{\partial o} \left( \frac{\partial h}{\partial u} \frac{\partial u}{\partial w} + \frac{\partial h}{\partial w} \right).$$

For the typical performance measure  $\ell_t = (1/2) \sum_i e_{i,t}^2$ , where  $e_t = y_t - o_t$  with  $y_t$  the target pattern employed at time  $t$ , we have  $\partial \ell / \partial o = -e'$  and hence

$$\frac{\partial \ell}{\partial w} = -e' M, \quad M = \frac{\partial o}{\partial w} = \frac{\partial h}{\partial u} S + \frac{\partial h}{\partial w}.$$

The updating rule for the sensitivities  $S = \partial u / \partial w$  can e.g. be obtained discretizing the corresponding continuous-time forward sensitivity equations which gives

$$S_{t+1} = \frac{\partial g}{\partial u} S_t + \frac{\partial g}{\partial w}.$$

As usual, the above can be combined with a variety of weight updating schedules. If the weights are updated at each time step, we obtain

$$\begin{aligned} \hat{e}_t &= y_t - h(\hat{u}_t, x_t; \hat{w}_t) \\ \hat{M}_t &= \frac{\partial h}{\partial u}(\hat{u}_t, x_t; \hat{w}_t) \hat{S}_t + \frac{\partial h}{\partial w}(\hat{u}_t, x_t; \hat{w}_t) \\ \hat{w}_{t+1} &= \hat{w}_t + \eta_t \hat{M}_t' \hat{e}_t \\ \hat{u}_{t+1} &= g(\hat{u}_t, x_t; \hat{w}_t) \\ \hat{S}_{t+1} &= \frac{\partial g}{\partial u}(\hat{u}_t, x_t; \hat{w}_t) \hat{S}_t + \frac{\partial g}{\partial w}(\hat{u}_t, x_t; \hat{w}_t), \end{aligned}$$

which is the on-line recurrent backpropagation algorithm considered in Kuan, Hornik, & White (1993). Clearly, if  $u$  is not present in the network, this algorithm simply reduces to the standard BP algorithm for feedforward networks. In the network (10),  $u_t = o_{t-1}$ . Hence,  $\hat{M}_t = \hat{S}_t$ , so that the equation for  $\hat{M}_t$  is redundant. The on-line recurrent BP algorithm then becomes (a more general variant of) the real-time recurrent backpropagation algorithm of Williams & Zipser.

All algorithms discussed thus far are based on (approximate) gradient descent and may thus exhibit very poor convergence behavior. For feedforward networks, Kuan & White (1993a) have shown that a stochastic Newton algorithm, which takes second-order derivatives into account as well, is computationally and statistically more efficient than the (gradient descent) BP algorithm and in fact asymptotically equivalent to the nonlinear least squares estimator under very general conditions. Their Newton algorithm is clearly motivated by the Newton method in numerical optimization and is analogous to that in the system identification literature; see e.g. Ljung & Söderström (1983). Analogously, the following Newton-type algorithm is a natural extension of the (real-time) recurrent BP algorithm (Kuan, 1993):

$$\hat{e}_t = y_t - h(\hat{u}_t, x_t; \hat{w}_t), \quad (11)$$

$$\hat{M}_t = \frac{\partial h}{\partial u}(\hat{u}_t, x_t; \hat{w}_t) \hat{S}_t + \frac{\partial h}{\partial w}(\hat{u}_t, x_t; \hat{w}_t) \quad (12)$$

$$\hat{H}_{t+1} = \hat{H}_t + \eta_t (\hat{M}_t' \hat{M}_t - \hat{H}_t), \quad (13)$$

$$\hat{w}_{t+1} = \hat{w}_t + \eta_t \hat{H}_{t+1}^{-1} \hat{M}_t' \hat{e}_t \quad (14)$$

$$\hat{u}_{t+1} = g(\hat{u}_t, x_t; \hat{w}_t) \quad (15)$$

$$\hat{S}_{t+1} = \frac{\partial g}{\partial u}(\hat{u}_t, x_t; \hat{w}_t) \hat{S}_t + \frac{\partial g}{\partial w}(\hat{u}_t, x_t; \hat{w}_t), \quad (16)$$

Again, if  $u$  is not present in the network, the recurrent Newton algorithm simply reduces to the Newton algorithm for feedforward networks. In contrast to the recurrent BP algorithm, the recurrent Newton algorithm contains an additional updating equation (13) which recursively updates the outer product of  $\hat{M}_t'$  to approximate the Hessian matrix and provide an approximate Newton direction for (14). This approximation is quite common in engineering, cf. e.g. Ljung & Söderström (1983); in particular, it makes it easier to ensure that the estimates of the Hessians remain positive definite and hence invertible throughout the algorithm.

There are some difficulties associated with this algorithm. First, it is not “local” because (14) involves matrix inversion. Second, it may not follow a correct search direction if  $\hat{H}_t$  is not a positive definite matrix. Let  $\nu_t = (1 - \eta_t)\eta_{t-1}/\eta_t$  and  $\hat{P}_t = \eta_{t-1} \hat{H}_t^{-1}$ . Then by a matrix inversion formula,

$$\hat{P}_{t+1} = \frac{1}{\nu_t} \left( \hat{P}_t - \hat{P}_t \hat{M}_t' (\hat{M}_t \hat{P}_t \hat{M}_t' + \nu_t I)^{-1} \hat{M}_t \hat{P}_t \right).$$

For the network with a single output,  $\hat{M}_t$  is a row vector, and hence

$$\hat{P}_{t+1} = \frac{1}{\nu_t} \left( \hat{P}_t - \frac{\hat{P}_t \hat{M}_t' \hat{M}_t \hat{P}_t}{\hat{M}_t \hat{P}_t \hat{M}_t' + \nu_t} \right),$$

which does not involve matrix inversion. For networks with multiple outputs, we can follow Bierman (1977) and compute  $\hat{P}_{t+1}$  using a sequence of single-output algorithms in which no matrix inversion is needed. Kuan (1993) thus proposes a modified Newton algorithm which contains (11), (12), (15), and (16) in the original version but substitutes the updating equations below for (13) and (14):

$$\bar{P}_{t+1}^{(0)} = \hat{P}_t, \quad (17)$$

$$\bar{P}_{t+1}^{(j)} = \bar{P}_{t+1}^{(j-1)} - \frac{\bar{P}_{t+1}^{(j-1)} \hat{m}_{j,t}' \hat{m}_{j,t} \bar{P}_{t+1}^{(j-1)}}{\hat{m}_{j,t} \bar{P}_{t+1}^{(j-1)} \hat{m}_{j,t}' + \nu_t}, \quad j = 1, \dots, \iota, \quad (18)$$

$$\bar{P}_{t+1} = \bar{P}_{t+1}^{(\iota)} / \nu_t, \quad (19)$$

$$\hat{P}_{t+1} = \begin{cases} \bar{P}_{t+1}, & \text{if } \bar{P}_{t+1} - \epsilon I \geq 0, \\ \bar{P}_{t+1} + \Delta_{t+1}, & \text{otherwise,} \end{cases} \quad (20)$$

$$\hat{w}_{t+1} = \hat{w}_t + \hat{P}_{t+1} \hat{M}_t' \hat{e}_t, \quad (21)$$

where  $\iota$  is the number of output units,  $\hat{m}_{j,t}$  is the  $j$ -th row of  $\hat{M}_t$ ,  $\epsilon$  is a small positive constant, and  $\Delta_{t+1}$  is chosen to make  $\hat{P}_{t+1} - \epsilon I \geq 0$ , i.e., a positive semidefinite matrix. Note that in (17)–(19),  $\bar{P}_{t+1}$  is updated as each output unit is added sequentially, and that (20) implements a correction ensuring  $\hat{P}_t$  to be a p.s.d. matrix.

In applications, one usually cannot guarantee a priori that the parameter estimates do not diverge to infinity. This is typically enforced by implementing some *projection device*  $\pi$ . I.e., if  $\theta$  is the vector of parameters updated by the algorithm under consideration (such that  $\theta = w$  in the recurrent backpropagation and  $\theta = [w', \text{vec}(P)']'$  in the Newton algorithm) and  $\Theta$  is a compact subset of the parameter space, then  $\pi(\theta) = \theta$  for  $\theta \in \Theta$  and  $\pi(\theta) \in \Theta$  for  $\theta \notin \Theta$ . The actual algorithm is then obtained by applying  $\pi$  to the estimates  $\hat{\theta}_t$  after each updating step. In practice, one would of course like to choose truncation bounds large enough so that the behavior of the algorithm is virtually unaffected by the projection device. However, this is not always possible, as suggested by the following convergence analysis.

## 4 Convergence Analysis

A rigorous and satisfactory analysis of the behavior of the learning algorithms presented thus far under the presentation of a sequence of learning patterns is clearly extremely hard, if not impossible. Suppose that the patterns are drawn at random. Then a learning algorithm generates a sequence of estimates  $\hat{\theta}_t$  according to some stochastic discrete-time system. In typical cases, it is possible to relate the analysis of this system to the asymptotic



behavior of an associated deterministic continuous-time system, the so-called *associated ODE*, which is obtained by “suitably averaging over all patterns”. Such a relation can be established for small constant learning rates  $\eta$  in the sense of weak convergence of random processes as  $\eta \rightarrow \infty$  (Kushner, 1984) as well as for learning rates  $\eta_t$  which decay to zero at a suitable rate (Kushner & Clark, 1978). The latter methodology has been used in particular to analyze the standard BP and Newton algorithms for feedforward networks, see e.g. Kuan & White (1993a). In the following, a general theorem due to Kuan & White (1993b) is given and applied to the recurrent BP and Newton algorithms.

Consider a general learning algorithm of the form

$$\begin{aligned}\hat{\theta}_{t+1} &= \pi(\hat{\theta}_t + \eta_t Q(\hat{r}_t, z_t, \hat{\theta}_t)), \\ \hat{r}_{t+1} &= \rho(\hat{r}_t, z_t, \hat{\theta}_t),\end{aligned}\tag{22}$$

where  $\pi$  is some suitable projection device on the parameter space and  $z$  is an exogeneous variable (i.e., the sequence of training patterns  $z_t$  does not depend on  $\theta$ ). Formally, all algorithms considered in this paper are of the above form, as the patterns  $z$  could be continuous-time functions of length  $L(z)$ . In what follows, we shall however assume that  $z$  is a finite-dimensional vector; this still covers all cases where the patterns are finite-length temporal sequences.

For a continuous vector field  $\nu(\cdot)$  let the vector field  $\bar{\pi}[\nu(\cdot)]$  be

$$\bar{\pi}[\nu(\theta)] = \lim_{\epsilon \rightarrow 0} \frac{\pi(\theta + \epsilon \nu(\theta)) - \theta}{\epsilon}, \quad \theta \in \Theta;$$

this technical structure is used to characterize the limiting ODE when the projection device  $\pi$  is imposed. Let

$$\tilde{\theta}^0(\tau) = \left( \frac{\tau_{t+1} - \tau}{\eta_t} \right) \hat{\theta}_t + \left( \frac{\tau - \tau_t}{\eta_t} \right) \hat{\theta}_{t+1}, \quad \tau \in [\tau_t, \tau_{t+1}),$$

where  $\tau_0 = 0$  and for  $t \geq 1$ ,  $\tau_t = \sum_{i=0}^{t-1} \eta_i$ , and let the left shifts  $\tilde{\theta}^t$  of  $\tilde{\theta}^0$  be given by

$$\tilde{\theta}^t(\tau) = \tilde{\theta}^0(\tau_t + \tau).$$

In other words,  $\tilde{\theta}^t$  is the process obtained by linearly interpolating the sequence  $\hat{\theta}_t, \hat{\theta}_{t+1}, \dots, \hat{\theta}_{t+k}, \dots$  at the time knots  $0, \eta_{t+1}, \dots, \eta_{t+1} + \dots + \eta_{t+k}, \dots$

Consider the following conditions.

[A1]  $\{z_t\}$  is a sequence of bounded, i.i.d. random variables.

[A2] The function  $Q$  is continuously differentiable of order one in all arguments.

[A3] The function  $\rho$  is continuously differentiable of order two in all arguments. Also, for each  $(z, \theta)$ ,  $\rho$  is a contraction mapping in the first argument  $r$ , i.e.,  $|\rho(r_1, z, \theta) - \rho(r_2, z, \theta)| \leq c|r_1 - r_2|$  with  $c < 1$ .

[A4]  $\{\eta_t\}$  is such that  $\sum_t \eta_t = \infty$  and  $\sum_t \eta_t^2 < \infty$ .

[A5] For each  $\theta \in \Theta$ ,  $\bar{Q}(\theta) := \lim_{t \rightarrow \infty} \mathbb{E}(Q(r_t(\theta), z_t, \theta))$  exists, where  $\Theta$  is the image of  $\pi$  and  $r_t(\theta)$  is generated by the recursion  $r_{t+1} = \rho(r_t, z_t, \theta)$ .

These conditions are stated for convenience; formal statements can be found in Kuan, Hornik & White (1993) or Kuan & White (1993b). The i.i.d. assumption in [A1] is in fact much stronger than needed; in particular, the result below continues to hold when  $\{z_t\}$  is a stationary and ergodic sequence or a weakly dependent sequence. The conditions [A2] and [A3] are “smoothness” conditions; some of the continuous differentiability conditions can be replaced by the Lipschitz continuity conditions. The condition [A4] specifies the order of learning rate; for example  $\eta_t$  of order  $1/t$  satisfies this condition. Finally, [A5] is needed to define the associated ODE. Note that the other conditions also imply that  $\bar{Q}$  is continuous.

The following result is established in Kuan & White (1993b).

**THEOREM.** *Suppose that [A1]–[A5] hold.*

- (a) *Let  $\tilde{\theta}(\cdot)$  be the limit of a convergent subsequence of  $\{\hat{\theta}^t(\cdot)\}$ . Then with probability one (w.p.1),  $\tilde{\theta}(\cdot)$  satisfies the ODE  $\dot{\theta} = \pi[\bar{Q}(\theta)]$ .*
- (b) *Let  $\Theta^*$  be the set of locally asymptotically stable equilibria in  $\Theta$  for the ODE in (a) with domain of attraction  $\mathcal{D}(\Theta^*)$ . If  $\Theta \subseteq \mathcal{D}(\Theta^*)$ , then  $\hat{\theta}_t \rightarrow \Theta^*$  w.p.1.*
- (c) *If  $\Theta \not\subseteq \mathcal{D}(\Theta^*)$ , but  $\hat{\theta}_t$  enters a compact subset of  $\mathcal{D}(\Theta^*)$  infinitely often w.p.1, then  $\hat{\theta}_t \rightarrow \Theta^*$  w.p.1. If, further,  $\Theta^*$  contains only finitely many points, then  $\hat{\theta}_t \rightarrow \theta^*$  w.p.1 for some  $\theta^* \in \Theta^*$ . This convergence is conditional on the initial value of the algorithm and the realization of  $\hat{\theta}_t$ .*

Part (a) of this theorem says that  $\hat{\theta}_t$  essentially follows the solution path of the ODE  $\dot{\theta} = \bar{Q}(\theta)$  in  $\Theta$ . Note that the projection device is not effective unless  $\theta$  is on the boundary of  $\Theta$  and the vector field  $\bar{Q}(\theta)$  points outside of  $\Theta$ . Then by the local asymptotic stability of the ODE,  $\tilde{\theta}(\cdot)$ , hence  $\hat{\theta}_t$ , converges to the set of locally asymptotically stable equilibria (part (b) and (c)). If there are only finitely many such equilibria, the last part of the theorem asserts that  $\hat{\theta}_t$  converges to one of them; cycling between equilibria is not possible.

By letting  $z_t = [x'_t, y'_t]'$  and  $r_t = [u'_t, \text{vec}(S_t)']'$ , we can apply the above theorem to the recurrent BP algorithms of Williams & Zipser (1989) and Kuan, Hornik & White and to the Newton algorithm of Kuan (1993). Hence, the convergence behavior of these algorithms can be obtained as corollaries. In these applications, it is readily seen that  $\bar{Q}(\theta) = 0$  implies the first order conditions for the minimization of the asymptotic mean square error  $E(\theta) = \lim_{t \rightarrow \infty} \mathbb{E} \sum_i (y_{i,t} - o_{i,t}(\theta))^2$ . Hence, a locally asymptotically stable equilibrium of the associated ODE is just a local minimum of  $E$ . From the above theorem, we can thus infer that these algorithms converge to a local asymptotic MSE minimizer with probability one.

The “smoothness” conditions imposed here are not very restrictive; most of the network activation functions adopted in applications, such as the logistic and hyperbolic tangent functions, satisfy these conditions. On the other hand, the contraction mapping condition in [A3] is crucial and is *not* satisfied automatically. In fact, it is easily seen that the underlying discrete-time system  $r_{t+1} = \rho(r_t, z_t, w)$  could “explode” if  $\rho$  is not a contraction mapping in  $r$ , because the effects of  $z_t$  could accumulate very rapidly. Even if  $\rho$  is a bounded (or squashing) function, this condition is still needed; otherwise,  $r_t$  could approach the upper or lower bound of  $\rho$  within a very short period of time, which would “exaggerate” the behavior of the internal states and invalidate the learning results. If  $\rho$  is a contraction mapping, then the system implements an exponentially forgetting memory of the data sequence and is essentially well-behaved. In fact, we notice that for the recurrent backpropagation and Newton algorithms, the contraction mapping property is (essentially equivalent to) requiring that all the eigenvalues of  $\partial g / \partial u$  be less than one in absolute value (uniformly in  $x$  and  $w$ ), which not only ensures proper behavior of the sequence of system states, but also is necessary for the stability of the forward sensitivity system, as has already been pointed out by Alneida (1987).

As an example, let us again consider the Jordan and Elman networks where

$$o_t = h(C\sigma(Au_t + Bx_t))$$

and  $u_t$  is the vector of lagged output or hidden layer activations, respectively. Let  $M_h$  and  $M_\sigma$  be the sup of the derivatives of the (usually identical) output and hidden layer activation functions, respectively. It is readily seen that the contraction mapping property for the Jordan network is satisfied when

$$\sum_j |c_{ij}| |a_{jk}| < (M_h M_\sigma)^{-1}$$

for all  $i$  and  $k$ . Similarly, we find that for the Elman network, it suffices that

$$\|A\|_F < 1/M_\sigma,$$

where  $\|A\|_F$  is the Frobenius norm of  $A$  ( $\|A\|_F^2$  is the trace of  $A'A$ , i.e. the sum of the squares of the entries of  $A$ ).

To ensure proper convergence behavior, the connection weights must be suitably constrained during the learning process. Note that in the Jordan network this is a restriction on the hidden-to-output weights and the internal feedback weights simultaneously. In the Elman network, however, only recurrent connections are subject to the constraint so that the feedforward part of the network is not affected. Thus, as far as the representation capability of a network is concerned, the Elman network is clearly superior since less network connection weights are subject to constraints. It is straightforward to verify that some sufficient conditions ensuring the contraction mapping property in the Elman network are:

- $|a_{ij}| < 4/\nu$  for all  $i, j$  if each  $\sigma_i$  is the logistic function,
- $|a_{ij}| < 1/\nu$  for all  $i, j$  if each  $\sigma_i$  is the hyperbolic tangent function,

where  $\nu$  is the number of hidden units.

These restrictions are practically important, as demonstrated in the simulation results of Kuan, Hornik & White (1993) and Kuan (1993); some of their results are summarized in Table 1. These simulations find that

- MSE of the Newton algorithm with the constraint
- < MSE of the Newton algorithm without the constraint
- < MSE of the BP algorithm with the constraint
- < MSE of the BP algorithm without the constraint.

In fact, the MSEs of the BP algorithm without the constraint are much larger than those of other algorithms; the Newton algorithm without the constraint behaves unstably and may produce very large errors during the learning period. On the other hand, both the BP and Newton algorithms with the constraint are well behaved, but the Newton algorithm results in much lower MSE and converges much faster than the BP algorithm. An interesting finding is that adding more hidden units need not result in lower MSE if a learning algorithm is used without the constraint, whereas the MSEs of the algorithm with the constraint typically decrease when the number of hidden units increases. This suggests that the “learned” network resulted from an algorithm without the constraint could be very misleading.



**Table 1.** Summary of Simulation Results.

Model	$\nu$	Newton with Const.		Newton w/o Const.		BP with Const.		BP w/o Const.	
		Average MSE	Last MSE	Average MSE	Last MSE	Avg. MSE	Last MSE	Avg. MSE	Last MSE
NL MA	4	N/A	N/A	N/A	N/A	1.280	1.253	1.332	1.307
	5	N/A	N/A	N/A	N/A	1.273	1.241	1.338	1.306
	6	N/A	N/A	N/A	N/A	1.268	1.229	1.339	1.304
BL 1	4	N/A	N/A	N/A	N/A	1.399	1.366	1.456	1.429
	5	N/A	N/A	N/A	N/A	1.392	1.351	1.449	1.416
	6	N/A	N/A	N/A	N/A	1.375	1.325	1.462	1.423
BL 2	4	1.825	1.824	1.900	1.839	2.181	2.154	2.547	2.533
	5	1.820	1.811	1.902	1.852	2.139	2.111	2.579	2.564
	6	1.802	1.789	1.924	1.867	2.109	2.078	2.634	2.615
HM	4	0.125	0.125	0.176	0.162	0.329	0.324	0.491	0.488
	5	0.101	0.098	0.159	0.142	0.324	0.319	0.527	0.524
	6	0.079	0.079	0.152	0.158	0.302	0.296	0.536	0.530

The networks are Elman networks with 2 input and  $\nu$  hidden units and a single output unit; inputs and targets at time  $t$  are  $[y_{t-2}, y_{t-1}]'$  and  $y_t$ , respectively, with the data generating processes for  $\{y_t\}$  being:

$$\text{NL MA (nonlinear MA)} \quad y_t = -0.3\epsilon_{t-1} + 0.2\epsilon_{t-2} + 0.4\epsilon_{t-1}\epsilon_{t-2} - 0.25\epsilon_{t-1}^2 + \epsilon_t,$$

$$\text{BL 1 (bilinear 1)} \quad y_t = 0.5 - 0.4y_{t-1} + 0.4y_{t-1}\epsilon_{t-1} + \epsilon_t,$$

$$\text{BL 2 (bilinear 2)} \quad y_t = 0.4y_{t-1} - 0.3y_{t-2} + 0.5y_{t-1}\epsilon_{t-1} + \epsilon_t,$$

$$\text{HM (Hénon map)} \quad y_t = 1 + 0.3y_{t-2} - 1.4y_{t-1}^2, \quad y_0 = -u, y_{-1} = 0.5u,$$

where the  $\epsilon_t$  are independent  $N(0, 1)$  and  $u$  is uniform on  $[0, 1]$ .

## References

- Almeida, L. B. (1987). A learning rule for asynchronous perceptrons with feedback in a combinatorial environment. In *Proceedings of the IEEE First International Conference on Neural Networks* (II: 609–618). San Diego: SOS Printing.
- Baldi, P. (1993). Gradient descent learning algorithms overview: a general dynamical systems perspective. *IEEE Transactions on Neural Networks*, to appear.
- Bierman, G. J. (1977). *Factorization Methods for Discrete Sequential Estimation*. New York: Academic Press.
- Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, **14**, 179–211.
- Gherry, M. (1989). A learning algorithm for analog, fully recurrent neural networks. In *International Joint Conference on Neural Networks* (II: 643–644). IEEE Press.
- Jordan, M. (1986). *Serial order: A parallel distributed processing approach*. Report ICS 8604, Institute of Cognitive Science, University of California at San Diego.
- Jordan, M. (1992). Constrained supervised learning. *Journal of Mathematical Psychology*, **36**, 396–425.
- Kuan, C.-M. (1993). *A recurrent Newton algorithm and its convergence properties*. BEBR Working Paper 93–0139, College of Commerce, University of Illinois at Urbana-Champaign.
- Kuan, C.-M., Hornik, K., & White, H. (1993). A convergence result for learning in recurrent neural networks. *Neural Computation*, to appear.
- Kuan, C.-M., & White, H. (1993a). Artificial neural networks: An econometric perspective. *Econometric Reviews*, to appear.
- Kuan, C.-M., & White, H. (1993b). Adaptive learning with nonlinear dynamics driven by dependent processes. *Econometrica*, to appear.
- Kushner, H. J. (1984). *Approximation and weak convergence methods for random processes*. Cambridge, MA: MIT Press.
- Kushner, H. J., & Clark, D. S. (1978). *Stochastic approximation methods for constrained and unconstrained systems*. New York: Springer-Verlag.



- Ljung, L., & T. Söderström (1983). *Theory and practice of recursive identification*. Cambridge, MA: MIT Press.
- Magnus, J. R., & Neudecker, H. (1988). *Matrix differential calculus*. New York: John Wiley & Sons.
- Norrod, F. E., O'Neill, M. D., & Gat, E. (1987). Feedback-induced sequentiality in neural networks. In *Proceedings of the IEEE First International Conference on Neural Networks* (II: 251–258). San Diego: SOS Printing.
- Pearlmutter, B. (1989). Learning state space trajectories in recurrent neural networks. *Neural Computation*, **1**, 263–269.
- Pearlmutter, B. (1990). *Dynamic recurrent neural networks*. Technical Report CMU-CS-90-196, Computer Science Department, Carnegie-Mellon University.
- Pineda, F. J. (1987). Generalization of back-propagation to recurrent neural networks. *Physical Review Letters*, **59**, 2229–2232.
- Robinson, A. J., & Fallside, F. (1987). Static and dynamic error propagation networks with application to speech coding. In Dana Z. Anderson (ed.), *Neural Information Processing Systems* (pp. 632–641). New York: American Institute of Physics.
- Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representations by error propagation. In Rumelhart, D. E., & McClelland, J. L. (eds.), *Parallel Distributed Processing*. Cambridge, MA: MIT Press.
- Sun, G.-Z., Chen, H.-H., & Lee, Y.-C. (1992). Green's function method for fast on-line learning algorithms of recurrent neural networks. *Advances in Neural Information Processing*, **4**, 333–340. Morgan Kaufmann.
- Toomarian, N., & Barhen, J. (1991). Adjoint-functions and temporal learning algorithms in neural networks. *Advances in Neural Information Processing*, **3**, 113–120. Morgan Kaufmann.
- Williams, R. J., & Zipser, D. (1989a). A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, **1**, 270–280.
- Williams, R. J., & Zipser, D. (1989b). Experimental analysis of the real-time recurrent learning algorithm. *Connection Science*, **1**, 87–111.













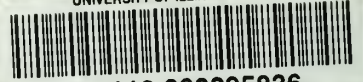
HECKMAN  
BINDERY INC.



JUN 95



UNIVERSITY OF ILLINOIS-URBANA



3 0112 060295836